Komplexitätstheorie 2004
Problemset 1

November 4, 2004

DIMACS Graph Format (.col)

Files in the DIMACS standard graph format are ASCII files which are organized in a line-by-line fashion:

- Lines which contain a **comment** start with a **c**.
- The **problem line** must appear exactly once within a file. It must be the first non-comment line. It must have the following form: **p edge |V| |E|** where the latter two are the number of nodes and edges respectively.
- **Edges** are described with a line of the form **e u v** where **u** and **v** are the two adjacent nodes. A graph file never contains **e u v** and **e v u** at the same time. u and v are taken from $1 \ldots |V|$.
- **Disclaimer:** The DIMACS format comes with more line types, however, we need only the mentioned line types.

Note, that the first character of a line (which indicates the line type) must be the first character of the line with no leading white-spaces.
In the following, you will have to parse such files. However, all programs which parse .col files **can behave arbitrarily** on syntactically incorrect input files.

Problem 1: 2Coloring Solver

Write a program which takes a DIMACS graph description via stdin and exits with exit code 0 if and only if the described graph is two-colorable.

Note: An efficient solution exits.

Problem 2: 3Coloring Solver

Write a program which takes a DIMACS graph description via stdin and exits with exit code 0 if and only if the described graph is three-colorable.

Note: Do not expect to find an efficient solution.

You are not allowed to use this solution as solution for problem 1.

Problem 3: k – Clique Solver

Write a program which takes a DIMACS graph description via stdin, and exits with exit code 0 if and only if the described graph has a k-clique.

The value of k must be specified in the file with a line of the following form **c required clique size k** which must be the first line in the file.

Note: Do not expect your solution to be efficient for arbitrary k and arbitrary graphs.

DIMACS CNF-Format (.cnf)

The DIMACS CNF-files are again ASCII-files which are organized line-wise:

- **Comments** start with **c**.
- The **problem line** must appear exactly once as the first non-comment line. Its format is **p cnf |V| |C|** where $|V|$ is the number of variables and $|C|$ is the number of clauses in the instance.
- **Clauses** make up the rest of the file (disregarding comments). A clause with positive literals over the variables $p_1, \ldots, p_k$ and with negative literals over the variables $n_1, \ldots, n_l$ is represented by a line $\mathbf{p_1}, \ldots, \mathbf{p_k}, -\mathbf{n_1}, \ldots, -\mathbf{n_l}$ where p_i and n_i are taken from $1, \ldots, |V|$.

Note, that the first character of a line (which indicates the line type) must be the first character of the line with no leading white-spaces.
As in the case of .col files, you are **not required** to write a parser which **reacts sensibly on incorrect input files**.

Problem 4: 2Sat Solver

Write a program which takes a DIMACS CNF description on stdin and exits with exit code 0 if and only if the described instance is a satisfiable 2SAT-instance, i.e., it must be satisfiable and each clause must contain either one or two literals.

Note: An efficient solution exits.

Problem 5: Sat Solver

Write a program which takes a DIMACS CNF description on stdin and exits with exit code 0 if and only if the described instance is satisfiable.

Note: Do not expect to find an efficient solution.

You are not allowed to use this solution as solution for problem 4.

Problem 6: Generalize: Hard vs. Easy Problems

- Some of the above problems did not have an efficient solution. Other did have such a solution. Can you make some general remarks? Can you find common characteristics of the hard problems?
- What happens, if you fed a solver for a hard problem with an easy instance, e.g., solve a 2Coloring-instance with your general solver (if you not handle the special case explicitly)?

Problem 7: Proving Membership within NP

- Assume, that your solver does not only decide the problem at hand buy also outputs a solution, for example, that a Sat-solver outputs a satisfying assignment.
  How hard is it to **check that the solution is valid**? Can you give efficient check-algorithms for Sat, 3Coloring, and Clique?

  Describe the algorithms in pseudo code.
- Prove that Sat, 3Coloring, and Clique are in **NP**!

Problem 8: Sat $\leq$ 3Sat

Write a preprocessor which takes a .cnf file as input on stdin, and outputs a .cnf file on stdout, such that the output instance contains no clause with more than three literals and is satisfiable if and only if the original instance was satisfiable.

Problem 9: Clique $\leq$ Sat *

Write a preprocessor which takes a graph description as input on stdin (k must be specified in the first line **c required clique size k**), and outputs a .cnf file on stdout such that the output is satisfiable if and only if the input graph had a k-clique.

If you wrote a Sat solver, experiment: pipe the result of the preprocessor into the Sat-Solver to decide your clique instances.

Problem 10: 3Sat $\leq$ Clique

Write a preprocessor which takes a .cnf file as input on stdin, and outputs a .col file on stdout such that the output has a k-clique if the input described a satisfiable instance. Your program can choose k freely but has to add a line of the form **c required clique size k** as the very first line of the file.

If the input instance contains clauses with more than three literals, your program can behave arbitrarily.

If you wrote a Clique solver, experiment: pipe the result of the preprocessor into the Clique-Solver to decide your clique instances.

If you also solved Problems 6 and 7 and wrote a Sat solver (or one of your colleges), try `sat_3sat | 3sat_clique | clique_sat | sat_solver`. Experiment.

Problem 11: Discuss the relative hardness

Consider the circle: `sat_3sat | 3sat_clique | clique_sat`. What can you say about the relative hardness of these problems? What happens if `sat_3sat | 3sat_clique | clique_sat` takes a lot of time?